RISC-V Encryptor – Progress Report

# 02/09/2022

I've gotten the core to run on the DE10-Lite and I ran C program on it. I've used the preexisting AI accelerator to multiply numbers. I'm now researching ways to implement AES. I read about two possible ways to implement AES:

1. Implementing the four stages
2. Using lookup tables as specified in [this paper](https://pdf.sciencedirectassets.com/277348/1-s2.0-S1875389212X00063/1-s2.0-S1875389212005822/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEGcaCXVzLWVhc3QtMSJGMEQCIFc4wQqU9AgyPceOWl5h%2BNXxrjvbNA%2FovU2O3zcEsMBuAiBkHviAJUNO9d8VCf72dk1zLiwSidhE9jvg%2Bzhnf5wU3SrVBAjv%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F8BEAUaDDA1OTAwMzU0Njg2NSIMNCfCICixS1NXDNl3KqkE2Uhy3CC8wUZaNGQgNqcePEZma13PjQC5q2eBnACvdq8N7QZWKbAiqZ4vW1RBstNBr2fP5q4ZXfsXkT3d99QBGTCtPRttRrs2XMzQgoVqwuCa6u398I1lRCulR%2Fh7oG%2Fn3sSzmurj7ffADZ78BMAY3a1bq%2FyKFDjX9%2BwCDU38FxYhpkBvIe486VEGpF6X9HkXxNwPCPxbN5P0NZqMOqPoX%2Fwx32CusHyC%2FqTIc%2B4axtsqWixzP7s0mkzMnH32UPNn8g8HfbfOKFagNFgVS3%2FEg4Tw%2FI5DARKQuj8wJdZW8zv%2B0F%2B5sqxxi2HsIhqlhW%2FDNdoBu4fPDeJggHKDcLwgW2noLgjxqiwpK1ItRNBWVmEWQ2HPjrY3qvuMoqH9PWXB5O9RiIbFeyJ4w2ne%2FMory2Di%2BVjXzIrw%2F5Wxk4Ll72aJ1qU0zSG7PJCc5ETjK2rAxesKVlvZyU5kWITTRmXXlR0AgXegsOqIyan0%2FOC9j9%2FXGnVCKvXGRx8cjxR1YP0w%2BNDio%2By1%2Bl%2BgpL%2F524DLz6ZFNsp3bTNU7%2FwC9t65FNd2nIXx2p62sdeXn0ohdSpE3L5HhhUPAzmaiklDLyqaJC976p02mfQ9MUbNAZ%2BHrmx2yidZ7rQyoj7xkjyYky4%2F3qjKMLpppPeIU95tLMC6hGscJbNHo4o2DuAD0PDWhX7N4vtYDPgZapE0PuIbgGS%2FfNW0HiuqVt8eOkcwASYjjpEkrLcg0uoy3zCJqN2YBjqqATpa2z7TN8enDChypoucPdCD3mD2bBmbrhMGalk8nws0lB2m%2FCYkdFGEVId5e%2BLkjB6XtiB0K%2FsJLHgtTevNGmGQKW9Dep9vgmNA9RLDbH8NewO3xDGQOpz4I3tuvVGyzbu4pkKmXWTxd%2B6j3wIZKRabTjsv%2FIbbRKDKq%2FMtlhcXBRSEBbRcrc4793YFIolkrwVypTR3D0wA53O4nVmG%2FfQk06gHNJYe%2FZG5&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20220906T145632Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTY7KJUTUJE%2F20220906%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=bf18e2aa6ef4c4788acdba4ef1252dc735f12b3775f434f762fa2ae21840df4d&hash=a22f313ffe89139680c0f65831238d3df678e1b8cb2541cde13929c14460ed15&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S1875389212005822&tid=spdf-61737145-0b55-4f23-8248-159ef4d35630&sid=5439c3f85e16c44bf76bcac5a4065666cb7fgxrqb&type=client&ua=4d575656560754545b07&rr=74680d475a5594dd)

I’m currently researching the lookup table method, as it sounds faster and easier to implement.

# 06/09/2022

I’ve started implementing AES encryption using lookup tables in python. Some parts of the implementation are based on the paper specification above, and some are based on [this go language library code](https://github.com/golang/go/blob/master/src/crypto/aes/block.go).

Code skeleton was taken from [this GitHub repository](https://github.com/boppreh/aes)

# 07/09/2022

I’ve successfully implemented AES in python using lookup tables. Currently, the code isn’t pretty, but it’s functional.

# 08/09/2022

I’ve made the preexisting AI accelerator bigger and allocated more memory to MMIO. It currently receives 2 128-bit values each as an array of 4 4-byte values. I’m testing a 4-wide vector multiplication unit where each scalar is one byte.

The code isn’t working. When multiplying the vector by the vector the FPGA reports that the multiplication result is 0.

# 11/09/2022

I’ve found multiple bugs when trying to setup a simulation, they are now fixed but the accelerator still isn’t working. I’m working on a simulation to find more bugs.

# 12/09/2022

Using the simulation, I’ve fixed the vector accelerator. See screenshot of the results below:  
Graphical user interface, application, table

Description automatically generated

# 13/09/2022

1. Fixed a few more bugs in the accelerator.
2. Built the key schedule module. Might need further testing
3. Started building the accelerator – mostly based on the python code I wrote.
4. To make sure that the accelerator works with the core, I’m using the 7seg display to show me the result of the vector multiplication. For some reason I can’t get the core to wait until the accelerator’s done flag is up and only then display the result.

# 14/09/2022

Regarding (4) from yesterday, I’ve found a bug in the core’s branch unit. If the branch isn’t taken, the branch unit sets the core’s instruction offset to 0 instead of 4. Took more than a day and a half of debugging to figure out.

I’ve also finished writing the first draft of code for the accelerator. Haven’t tested it yet.

# 18/09/2022

Fixed bugs in the encryption units.

# 19/09/2022

Finished the encryption unit. Also integrated the unit with the core. Now starting to expand current infrastructure to support two cores.

The problems that we need to solve to implement the dual-core design managing the reads and writes to memory and to MMIO devices.

# 20/09/2022

Added another port to program memory and data memory. Currently not working.

# 21/09/2022

Got the two cores working simultaneously. Only had one major bug – I’ve declared the Verilog arrays in the wrong order.  
Because the data memory is 1KB in size, I’ve changed core1’s initial stack pointer value to 512 to prevent conflicts. If both cores write to the same memory address at the same time, the value written by core0 will be the one saved.

Now I need to create a module that’ll regulate the access to the encryption unit. I’ll create a lock-mux hybrid module.

* The cores will be wired up to the lock-mux, which will in turn we connected to the encryption unit.
* If a core wants to use the encryptor, it’ll need to acquire the lock and only then use the encryptor. The lock-mux will pass the appropriate inputs to the encryptor.
* The Encryptor’s content (key, plaintext…) will be hidden from the other core while it’s in use.
* The lock will be acquired by writing to a memory address the value “1”. The core can validate that it has acquired the lock by reading from this address. The value that’ll be read from this address is the core’s index (0 or 1 in our case). Because the core that I have doesn’t support atomic operations, I’ve given priority to core0.

# 25/09/2022

I’ve finished writing the code for the lock-mux. I’ll test it via simulations.

# 28/09/2022

It appears that the simulator that I have (modelsim-altera) has some problems with the lock-mux having an unpacked array of packed arrays as input/output as I keep receiving errors that I can’t solve.

I’ve instead wired up the lock-mux to the cores and encryptor and I’m testing it directly on the FPGA, swallowing the long compilation times.

# 29/09/2022

For some reason, everything broke. I should have committed the code more frequently.  
I would “go back to basic” and retest everything, but the long compilation times make this unbareable and seem like a waste of time.

Instead, I’ve created a simple test involving the two cores, and I’ll figure out how to shorten the compilation time and how to load C programs without a full recompilation.

# 30/09/2022

Changing the data memory to RAM shortened the compilation times from 20 to 4 minutes and lowered LE utilization from about 80% to about 30%.

For some reason, the data ROM isn’t working properly.

# 31/09/2022

I’ve misused the ROM. I’ve configured it with 32-bit words and a depth of 2048 words, but didn’t realize this means that the address 0x1 is 32 bits after address 0x0. Some things still aren’t working.

# 01/10/2022

Things seem to not work for random reasons. For example: if some statement is inside an ‘if’ statement, the processor might not execute it.

I’ve verified that the processor does read the appropriate code.

I’m unable to look directly inside the processor’s memory, so I’ll use the 7-segment display and the switches for debugging purposes.

It appears that the cores are receiving instructions with a 1-cycle delay. Might be a problem in Quartus 17 (according to Google). Moving to Quartus 21.

This didn’t solve the problem.

# 02/10/2022

I’ve failed to understand that without proper design, the CPU won’t play nice with actual memory (as opposed to registers) and it seems that the CPU that I have doesn’t currently support memory. It appears proper memory support was removed at some point.

It would be too difficult to rebuild the CPU and probably out of the scope of this project.

After trying many different solutions (that didn’t work), I’ve settled on multiplying the memory’s clock frequency by 10.

I can now once again resume to testing the lock and encryption unit.

# 04/10/2022

I’ve found multiple places where I’ve used the wrong indexes, including in the data memory file. This caused that instead of using 2KB of data memory I used 256 bytes or so.

# Notes

## Dual-core

The program counter is initialized to 0xcc by default. This is the first instruction of main.

The problems that we need to solve to implement the dual-core design managing the reads and writes to memory and to MMIO devices.

For MMIO devices, I’ll implement a lock that the core will need to acquire to release. The lock should probably just be a MMIO bit that should only be accessed via xchng and the software interfaces for those devices should be revised.

For memory, I’ll add the option to read and write two values in one cycle. If the target address is the same, core0 will have priority. If the address being read is currently being written, I’ll return the new value.

Another problem is the stack. I’ll need to have a stack for each core.

It appears that there are two different memory types – program memory (code section) and data memory. The data memory is 1KB and the program memory is 8KB. Regarding the stack issue – the registers are set to 0 at startup. This means that I can reset the stack pointer in core1 to 512 and this will create two stacks – I need to hope they don’t cross into each other.